

TP1 - BackEnd JEE

Tom REDON

5 Octobre 2021

Contents

1	Introduction	3
2	Exercice 1	3
3	Exercice 2	4
4	Exercice 3	5
5	Exercice 4	6
6	Exercice 5	7
7	Conclusion	9

1 Introduction

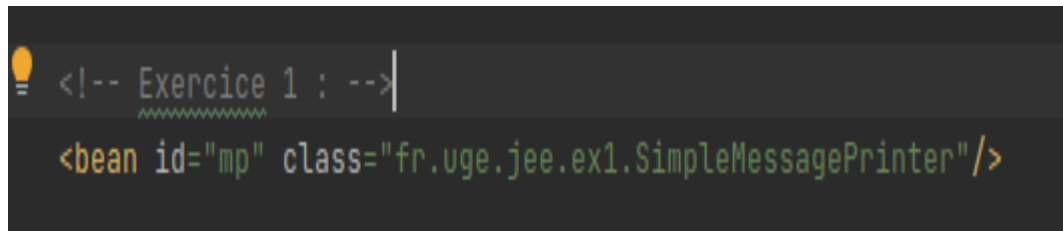
Le but de ce document est de décrire le cheminement de pensée qui s'est établi pendant le travail effectué sur ce TP.

Nous allons décrire exercice par exercice les décisions prises et les solutions établies.

Enfin, pour conclure, nous établirons les difficultés et les apprentissages que ce TP m'aura permis d'acquérir.

2 Exercice 1

Le premier exercice était assez trivial. Nous créons la classe "SimpleMessagePrinter" souhaitée une classe Application qui se charge d'accéder au fichier XML en l'identifiant comme un ApplicationContext et d'en appeler le bean établi pour obtenir l'objet et ainsi pouvoir appeler la méthode printMessage(). Le fichier XML contient la ligne suivante :



```
<!-- Exercice 1 : -->
<bean id="mp" class="fr.uge.jee.ex1.SimpleMessagePrinter"/>
```

Figure 1: Exercice 1

On identifiera donc par "mp" le bean à récupérer lors de l'appel de context.getBean().

3 Exercice 2

Nous reprenons le code de l'exercice précédent et nous le complétons en ajoutant une interface `MessagePrinter`, puis nous faisons en sorte que les classes `SimpleMessagePrinter`, `FrenchMessagePrinter` et `CustomizableMessagePrinter` implémentent cette interface.

Il nous devient donc possible de ne plus modifier le fichier `Application` pour switcher entre ces différents `Printer`. Il suffit d'identifier le `Printer` souhaité avec l'id de détection enregistré dans `Application`, soit "mp".

Vous pouvez donc voir comment le fichier XML s'organise pour répondre aux diverses questions. Les premières sont mises en commentaire dans l'image :

```
<!-- Exercice 2 : -->
<!-- <bean id="mp" class="fr.uge.jee.ex2.FrenchMessagePrinter"/> -->
<!-- Le constructeur sans paramètre est appelé pour CustomizableMessagePrinter avec : -->
<!-- <bean id="mp" class="fr.uge.jee.ex2.CustomizableMessagePrinter"/> -->

<bean id="smp" class="fr.uge.jee.ex2.SimpleMessagePrinter"/>
<bean id="fmp" class="fr.uge.jee.ex2.FrenchMessagePrinter"/>
<bean id="mp" class="fr.uge.jee.ex2.CustomizableMessagePrinter">
    <constructor-arg value="Hello from the config-2.xml"/>
</bean>
```

Figure 2: Exercice 2

Le code XML actif permet donc de désigner comme bean utilisé le bean du `CustomizableMessagePrinter`. Si on ne donne pas de `constructor-arg` lors des instructions du bean, le constructeur utilisé par défaut est celui sans paramètres. On a donc appris à passer un argument au constructeur dans la déclaration du bean.

4 Exercice 3

Nous reprenons le code de l'exercice précédent et nous le complétons avec une classe `CountMessagePrinter` qui implémente l'interface `MessagePrinter`. Cette classe contient donc un compteur qui incrémente les messages petit à petit. Lors du test de l'affichage donné en consigne, on peut se rendre compte que l'incrément continue même lorsque l'objet n'est plus désigné de la même façon. Ainsi, on peut deviner que le conteneur IoC met en place un Design Pattern Singleton. Ce Design Pattern permet de restreindre l'instanciation d'une classe à un seul objet.

```
<!-- Exercice 3 : -->
~~~~~

<bean id="smp" class="fr.uge.jee.ex3.SimpleMessagePrinter"/>
<bean id="fmp" class="fr.uge.jee.ex3.FrenchMessagePrinter"/>
<bean id="cmp" class="fr.uge.jee.ex3.CustomizableMessagePrinter">
    <constructor-arg value="Hello from the config-3.xml"/>
</bean>

<!-- Le scope de base est le Design Pattern Singleton -->
<bean id="printerServiceCount" class="fr.uge.jee.ex3.CountMessagePrinter" scope="prototype"/>
```

Figure 3: Exercice 3

On apprend ici à modifier le "scope" que peut donc avoir le conteneur IoC. En donnant le scope `prototype` en paramètre, on peut voir qu'un nouvel objet est instancié lors du deuxième appel à la fonction `context.getBean()`.

5 Exercice 4

Pour cet exercice, j'ai repris le code donné dans le cours. J'ai modifié les noms des livres et j'ai rajouté un bean désignant un troisième livre. Ensuite, le bean de la librairie contenait bien deux livres (Livre 1 et Livre 2).

Ensuite, on teste l'autowire "constructor". Cet autowire ajoute automatiquement TOUS les livres (les 3) dans la librairie. Voici le code XML établi :

```
<bean id="book1" class="fr.uge.jee.ex4.Book">
  <constructor-arg value="Harry Potter à l'école des sorciers"/>
  <constructor-arg value="00001"/>
</bean>
<bean id="book2" class="fr.uge.jee.ex4.Book">
  <constructor-arg value="Guerre et Paix"/>
  <constructor-arg value="00002"/>
</bean>
<bean id="book3" class="fr.uge.jee.ex4.Book">
  <constructor-arg value="Tests et essais"/>
  <constructor-arg value="00003"/>
</bean>
<bean id="library" class="fr.uge.jee.ex4.Library" autowire="constructor"/>
```

Figure 4: Exercice 4

Enfin, pour simplifier le code de la question précédente, cela ne me semblait pas possible pour l'exercice 3, je pense donc que l'exercice 5 était désigné. Nous verrons donc la réponse pour cette question dans le prochain exercice.

6 Exercice 5

Pour cet exercice, j'ai essayé d'écrire le code le plus maléable possible en ajoutant les interfaces nécessaires à un développement futur du projet d'OnlineShop.

Il existe donc les interfaces Service, qui est une interface étendue par les interfaces Delivery et Insurance. Cette interface force l'implémentation d'une méthode getDescription.

L'interface Delivery est implémentée par les classes StandardDelivery et DroneDelivery.

L'interface Insurance est, elle, implémentée par les classes ReturnInsurance et TheftInsurance.

Dans toutes les classes, on définit tous les setters pour les éléments.

On établit ensuite les Beans nécessaires pour obtenir l'affichage souhaité en ne faisant que des injections avec les setters. On obtient donc le code XML suivant :

```
<bean id="droneDelivery" class="fr.uge.jee.onlineshop.DroneDelivery" />
<bean id="standardDelivery" class="fr.uge.jee.onlineshop.StandardDelivery">
  <property name="delay" value="5"/>
</bean>

<bean id="returnInsurance" class="fr.uge.jee.onlineshop.ReturnInsurance">
  <property name="everyone" value="false"/>
</bean>

<bean id="onlineShop" class="fr.uge.jee.onlineshop.OnlineShop">
  <property name="name" value="Amazon"/>
  <property name="deliveryOptions">
    <set>
      <ref bean="droneDelivery"/>
      <ref bean="standardDelivery"/>
    </set>
  </property>
  <property name="insurances">
    <ref bean="returnInsurance"/>
  </property>
</bean>
```

Figure 5: Exercice 5 : Partie 1

Enfin, on a uniquement à modifier le fichier XML pour obtenir le second affichage souhaité. En se servant de ce que l'on a appris avec l'autowire dans l'exercice précédent, on obtient le code XML suivant :

```
<bean id="standardDelivery" class="fr.uge.jee.onlineshop.StandardDelivery">
  <property name="delay" value="999"/>
</bean>

<bean id="onlineShop" class="fr.uge.jee.onlineshop.OnlineShop" autowire="byType">
  <property name="name" value="AhMaZone"/>
  <property name="insurances">
    <set></set>
  </property>
</bean>
```

Figure 6: Exercice 5 : Partie 2 et Exercice 4 : dernière question

7 Conclusion

Pour conclure, j'ai trouvé ce TP très clair et bien guidé. En suivant les instructions données et en relisant le cours, je pense avoir réussi à réaliser les exercices demandés. J'ai rencontré un peu plus de difficultés lors de ma première utilisation des autowires, mais en ayant la possibilité de les utiliser, cela m'a paru encore plus clair. Enfin ce TP m'aura permis de commencer à apprendre à me servir des Beans de Spring, ainsi que leurs attributs et leur fonctionnement.